Mechatronics (Mechanical System Control): It's The Software!

David M. Auslander

Mechanical Engineering

University of California at Berkeley

Preamble

1. Don’t take the name “Mechatronics” too literally
2. The biggest value-added in mechatronics is in software
3. Mechanical system control:

   Then: Striving for complexity

   Now: More complexity than we can handle!

Mechanical Systems

- A long history of complexity in mechanical devices
- Modulation of power for delivery to a target
- Broadly applied to physical systems
  - Motion, thermal, fluid, chemical
- Thesis: Software has made this into a new game!

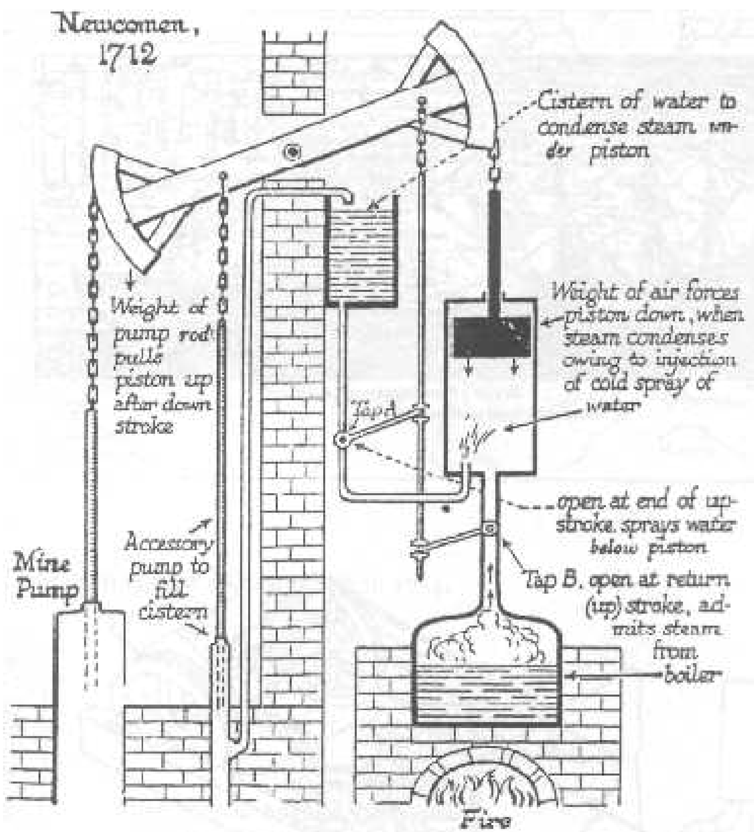
A Little History

- Computation in control of early machines
- Delivery of power – steam engines
- Complex pattern generation – Jacquard loom
- Brushed DC motor

Fairbottom Bobs

- Newcommen engine(~1760)
- http://www.ashton-under-lyne.com/bobs.htm
- Ashton under Lyne
- Pumped water from coal pits
- Photo ~1880

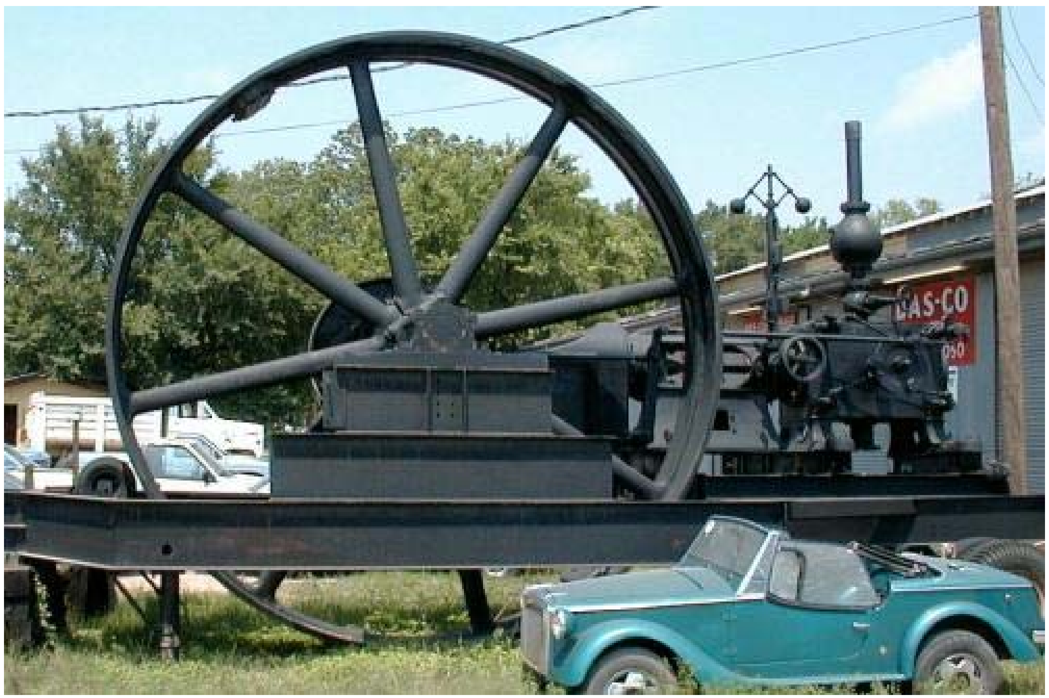
Newcomen Engine Control



DIAGRAMMATIC VIEW OF NEWCOMEN'S ATMOSPHERIC OR FIRE ENGINE (1712)

- www.technology.niagarac.on.ca/courses/tech238g/newcomen.htm
- Atmospheric steam engine
- Used water spray to condense steam in cylinder
- Control of valve based on walking beam position
- 1712, invented first usable steam engine

The Watt Governor



Courtesy of Vintage Saws

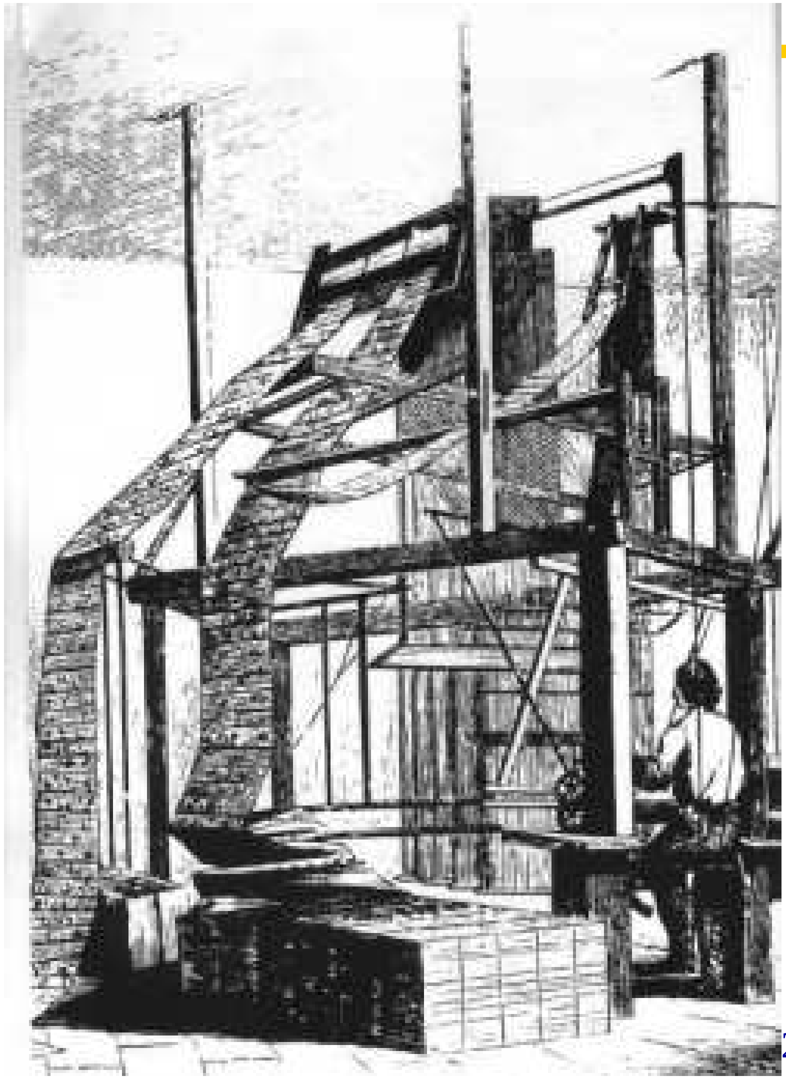
- http://www.vintagesaws.com/library/steam/steam.html
- For cotton mill
- 1856
- 100 HP, 30 RPM
- Note flyball (Watt) governor
- Smithville, Texas, USA

Closeup of Governor

Courtesy of Vintage Saws

- Note link that connects flyball to steam valve
- Major limitation of all classically controlled mechanical systems

Jacquard Loom



- http://www.digidome.nl/history.htm
- Punch card driven
- 1804
- Used to weave very complex patterns in silk

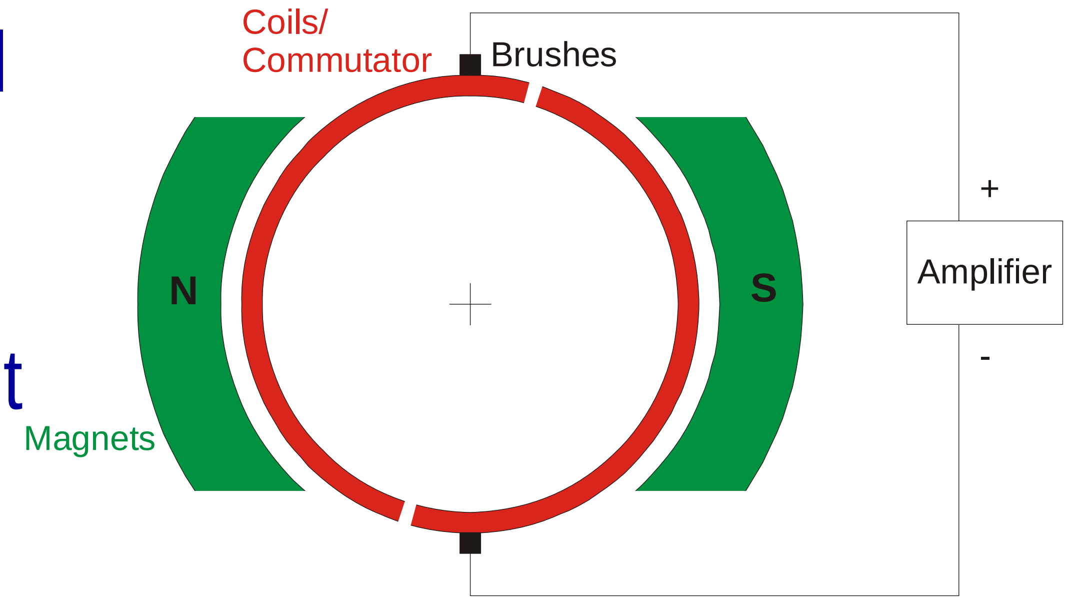
Silk Woven on a Jacquard Loom

- Silver and gold threads used for the pattern

Still In Use …

Classical Mechatronics – Brush vs. Brushless Motors

- Brush motor – classic mechanical system
- Rotor – coils
- Stator – permanent magnets
- Commutator computes



Complexity Limitation in Classical Mechanical Systems

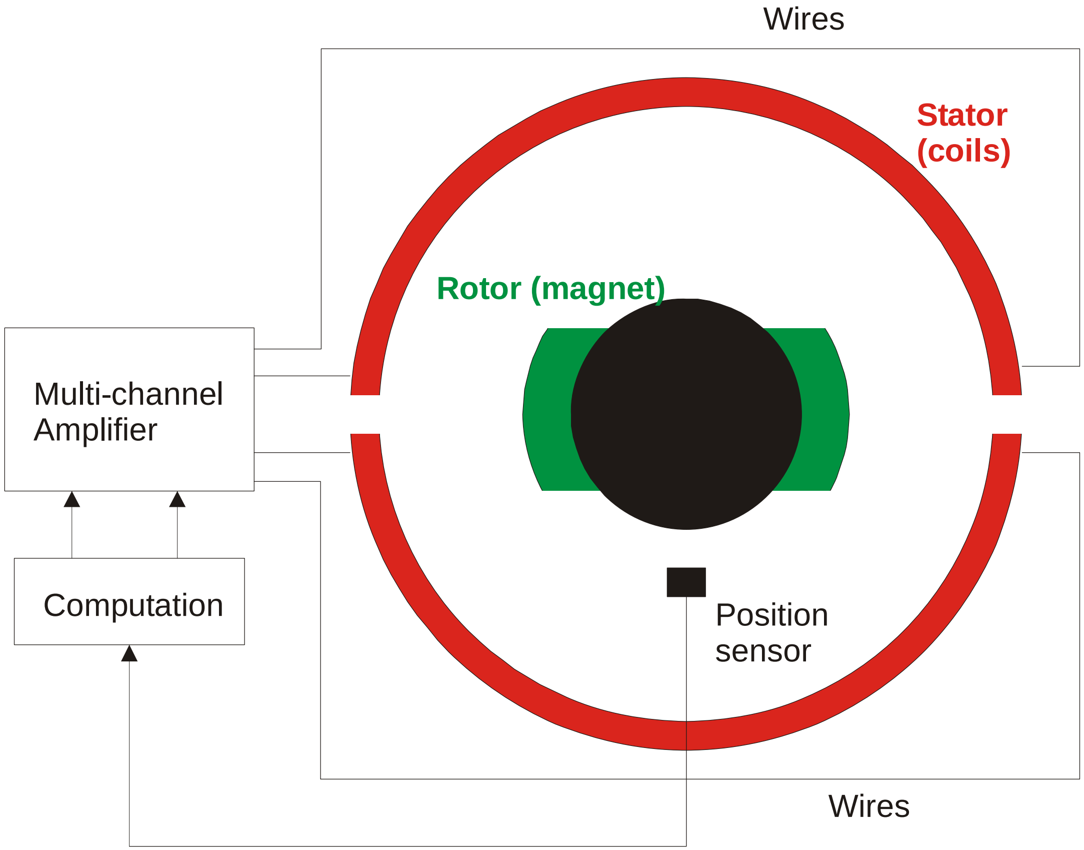
- No separation of sensing, computation, and power
- Example: flyball governer
  - Power to operate steam valve comes from flyball
  - Must have low-impedance path from main shaft all the way to the steam valve
  - Common physical medium (mechanical)
- Example: Brush motor, commutator

Enter Mechatronics

- Yaskawa Electric coined the term around 1970 to describe brushless motor technology
  - Trademark, then released
- Adding electronics made a completely different class of system
- Inconceivable in prior era

Brushless Motor

- Brushless – invert rotor and stator
- Measurement of rotor position needed to properly excite stator coils



Modern Mechatronics

- Add economic, compact computation
- "The synergetic integration of mechanical engineering with electronics and intelligent computer control in the design and manufacturing of industrial products and processes." (IEEE/ASME Transactions On Mechatronics, 1996)
- Or, "The application of complex decision-making to the control of physical systems"

What's Unique?

- The shorter definition focuses more strongly on the uniqueness of software-driven mechanical systems
- They can have control complexity beyond the wildest dreams of pre-computer engineers
- Control – interpreted broadly, not just feedback control

Enabling Technologies - I

- Amplification
  - Vacuum tube, Lee De Forrest, 1906
  - Flapper nozzle valve, pneumatic and hydraulic
- Enabled isolation of measurement, computation and actuation
  - Impedance mismatch vs. impedance match
  - Optimization of medium

Enabling Technologies - II
The Emergence of Software

- Process control was initial application
- Could afford very expensive, large hardware
- Improved productivity, reliability
- Microprocessor invention dramatically lowered cost-of-entry
- Now – cheapest way to control power modulation

Real Time Software

- Software is data reproducible
  - Successive program operation with same input data produces same output
- This is a defining property of computation and software – no error propagation!
  - Essence of digital systems: no complexity limit
- However, it is not generally time reproducible
- Example – timed loop with histogram

Same Program, Run Twice

Time (sec)	# less than	Time (sec)	# less than
2.0E-6	0	2.0E-6	0
4.0E-6	2640102	4.0E-6	2634222
8.0E-6	16972	8.0E-6	16968
1.6E-5	46	1.6E-5	65
3.2E-5	348	3.2E-5	362
6.4E-5	241	6.4E-5	230
1.28E-4	197	1.28E-4	214
2.56E-4	91	2.56E-4	100
5.12E-4	123	5.12E-4	160

Real Time for Mechatronic Control

- In brief, specifications (tolerances) for time reproducibility
- Each module of control software will have its own specs
- “Hard” (deadline) real time is not usually required
- Much of the activity is asynchronous preventing deterministic scheduling

Design Principle

- If any mechanical components are present to convey information consider replacing them with software (first choice) or electronics
- Examples
  - Brushes → brushless motor
  - Carburetor → fuel injection
  - Kinematic linkages, cams → motion profiles
  - Air dampers → variable speed motors

Design Context: The Unit Machine

- Domain of applicability
- Establish appropriate design methodology
- “The elements of a *unit machine* exchange physical power with each other or exchange material with little or no buffering”
- Unit machine too big ➔ can’t handle complexity
- Unit machine too small ➔ can’t optimize

Example: Semiconductor Mfg

Redefinition of the "unit machine" improved throughput by 3.5 times for a system similar to this!

Courtesy of Berkeley Process Control, Inc

Control Software for a Unit Machine

- Must have rapid access to all internal information
  - Sensors, actuators, states, commands, etc.
  - Rapid: fast enough to use in control loops
  - Can be used to optimize operation
- Information between unit machines usually simple commands, limited scope, slow

Unit Machine Examples

- Wafer handling robot
  - Commercially defined as unit machine
  - Often too simple
- Denver airport baggage handling
  - Too complex to be treated as unit machine
  - Defeated by complexity (my opinion!)
- Dynamic definition of unit machine in biology
  - Human gait change from walking to running

Industrial Experience: Complexity is the Problem

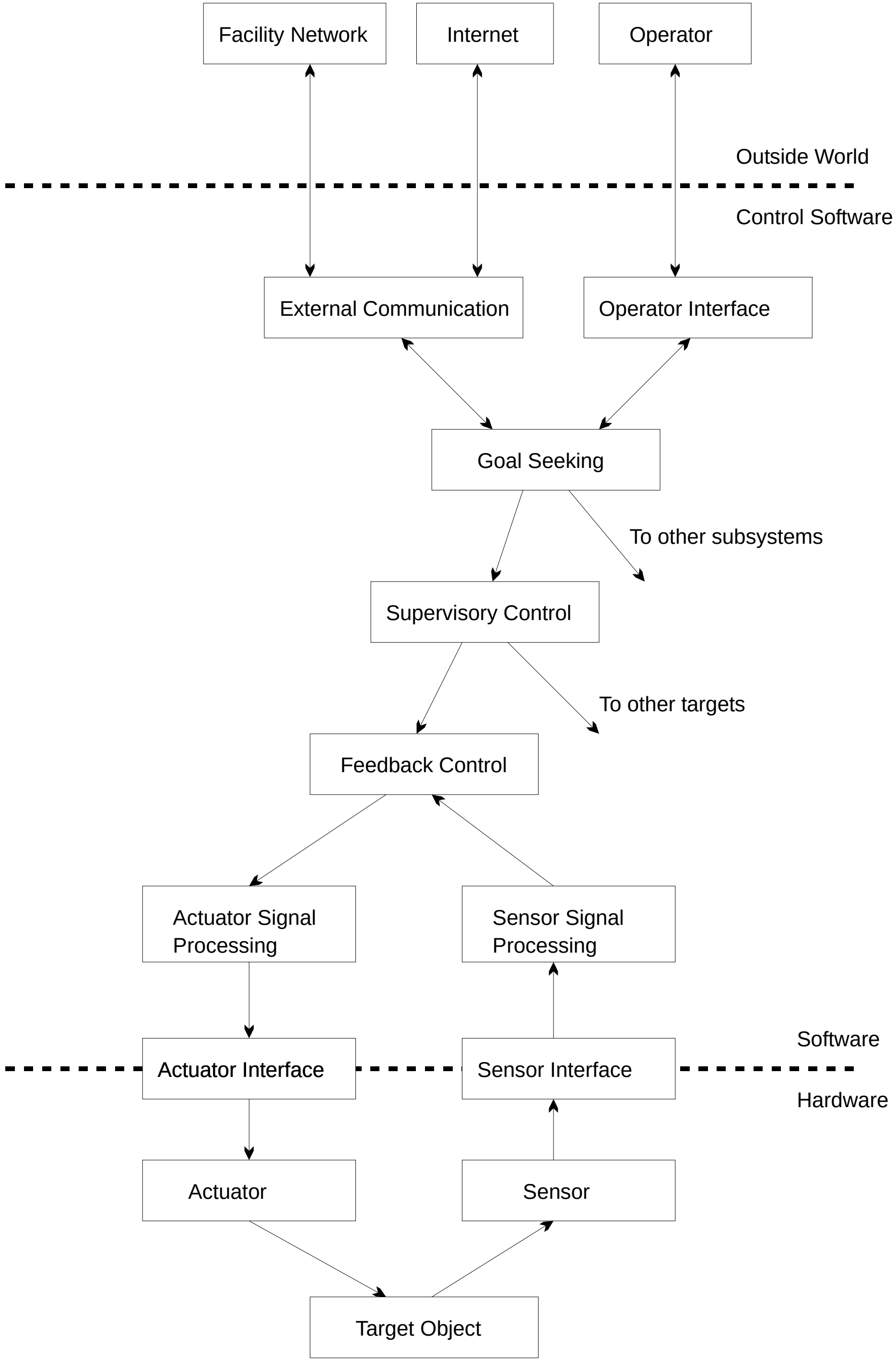
- Control (read software) is the largest cause of failure in complex manufacturing machines
- Failure to understand the consequences of complexity lead to unreliable operation, poor performance
- Mechanical engineers – don't understand software
- Software engineers – don't understand machines

Design Language

- A means of describing and documenting solutions; map easily to software
- More abstract than code
  - Most code is unreadable – even by the person that created it!
- Communication vehicle among all stakeholders
  - Engineering, manufacturing, marketing, maintenance, etc.

Tasks and State Machines

- Tasks – Simultaneously executing modules
- “A task is a well-defined responsibility” (American Heritage Dictionary)
- Suitable for complexity levels associated with unit machines
- Hierarchical organization
- Lowest level maps to mechanical system hardware
- Higher levels are goal oriented (next slide …)



Facility Network
Internet
Operator
Outside World
Control Software
External Communication
Operator Interface
Goal Seeking
To other subsystems
Supervisory Control
To other targets
Feedback Control
Actuator Signal Processing
Sensor Signal Processing
Software
Actuator Interface
Sensor Interface
Hardware
Actuator
Sensor
Target Object

Finite State Machines

- Internal task structure is finite state machine
- States consist of three sections
  - Entry – executed only after a transition
  - Action – execute always
  - Transition test
- Tasks and associated state machines provide a design model that is widely accessible as well as translatable into functioning software

Implementation Languages

- Desired properties in implementation languages
  - Portability
  - Clean syntax
  - Efficient footprint and operation
  - Well documented

Software Portability

- Primary factor in productivity/economics
- Development stages
- Production upgrades
- Processor generation time: 18 months or less
- Mechanical generation time: 5 – 20 years

Computational Implementation

- Clean separation between design and software implementation
- Focus on mapping design to software
  - Easily connect sections of software with design elements
- To the extent possible, weaken the dependence on language, OS, environment, hardware specifics

Cooperative Multitasking

- Most portable form of multitasking
- Requires one major stylistic restriction:
  - All code must be non-blocking
  - State machine fits this model very well
- “Universal” real time model –
  - If computer is fast enough, can meet all specs
  - Often true
- Otherwise, interrupts, priorities, etc. involve resource shifting
- See plenary talk by Michael Pont on this subject

Low/Medium Level Languages

- Object-oriented approach
- "Implementing two motors for position control within the program was a rather straightforward approach …" (Berkeley student)
- C, C++ and Java
- Java easier to learn, cleaner syntax, much better portability, but has performance problems
- C still the most widely used

High Level Languages

- Programming productivity – lines/day, regardless of language
- Therefore, use language requiring fewer lines
- Code generator (as in Matlab/Simulink) or embed into processor (e.g., Labview)
- More practical as processors get faster
- Still used more for development than production

Simulation

- Crucial step in design process but often skipped
- Conceptual and execution errors much more easily found than in real environment
- Takes initial effort to set up simulation
  - Engineers don’t want to spend the time!
- Dilemma: How to avoid rewriting control code for simulation?
- Major portability challenge

Simulation Environments

- C, C++, Java
  - Limited mathematical and plotting support
- Matlab/Simulink and other mathematical environments
  - DLL for control code or use code generator
- Ch
  - C for control code; C and limited C++ for simulation with rich math and graphics – easy to integrate C

Implementation Environments

- From lab to production, PC to microcontroller
- Real time
  - General purpose OS (Windows, etc.) ~1 second (can be used faster for demo, debug, but not reliable)
  - Real Time OS (RTOS - QNX, VxWorks, etc) sub-millisecond for regular tasks, microsecond for interrupts
  - Labview-RT – sub millisecond, shell must be Labview
  - Bare processor – microsecond
- Cooperative multitasking improves portability

Multiprocessor and Networking

- Networking becoming ubiquitous even in control systems
- Many network “standards”
- Ethernet gaining ground, but no winner yet
- Three network levels (at least!)
  - Sensor and actuator
  - Control processor
  - Factory (sometimes a cell level also)

Networking and Control Software

- Portability again the key
- Treat tasks as indivisible network components
- Abstract intertask communication
  - That is, custom application layer for intertask communication
- Allows for isolation of actual network protocols

Future Directions

- Moore's law still holds, but direction has changed
  - Multi-core processors rather than more powerful
  - Only likely to impact high-end systems in near future
- FPGA (field programmable gate array)
  - New processor frontier
  - Fully parallel
  - Usually viewed as circuit element but complexity has increased so now looks like processor with software

Lessons Learned

- Managing complexity is the challenge
- Modularity is the primary tool
  - Unit machine for hardware design
  - Tasks, state machines for software design
- Too much modularity limits the amount of global optimization that can be done
- Too little leads to unpredictable behavior and cost
- Therefore, err on the side of too much modularity